queue < type > q;

1. q.push(element)

2. q.pop()

3. q.front()

4. q.back()

5. q.empty() (!)

6. q.size()

7. q.swap(q2)

----------------------------------------------------------------------------------------------------------------------------------

stack <int> sk;

1. sk.empty() (!)

2. sk.pop()

3. sk.push(element)

4. sk.top()

5. sk.size()

6. sk.swap(sk2)

----------------------------------------------------------------------------------------------------------------------------------

priority\_queue < type , vector < type > , cmp > pq;

// cmp : //opposite

struct cmp {

bool operator()(const type &aa,const type &bb) {

return aa.ff < b.ff;

}

};

1. pq.push(element)

2. pq.pop()

3. pq.top()

4. pq.empty() (!)

5. pq.size()

6. pq.swap(pq2)

----------------------------------------------------------------------------------------------------------------------------------

string s;

1. operator " + "

2. operator " = "

3. operator " [] "

4. s.size()

5. s.substr(pos) s.substr(pos,length)

6. s.swap(s2)

7. s.find(string) s.find(string,pos) //use size\_t

//0-based indexing // k!=string::npos

8. s.replace(pos,len,string)

9. we can sort a string

----------------------------------------------------------------------------------------------------------------------------------

vector < type > vi;

1. vi.assign(val,no.of) vi.assign(it first,it last) vi.assign(arr , arr+N)

2. vi.clear()

3. vi.empty() (!)

4. vi.push\_back()

5. vi.pop\_back()

6. vi.erase(it) vi.erase(it first,it last)

7. vi.front()

8. vi.back()

9. vi.insert(it pos,val,no.of) vi.insert(it pos,it first,itlast)

vi.insert(it pos,arr, arr+N)

10. operator " = " ( vector1 = vector2 )

11. operator " [] " (vi[i])

12. vi.begin() vector<typ>::iterator it

13. vi.end() (!)

14. vi.rbegin() vector<typ>::reverse\_iterator it

15. vi.rend()

16. vi.size()

17. vi.swap(vii)

18. lower\_bound(vi.begin(),vi.end(),element)

19. upper\_bound(vi.begin(),vi.end(),element)

bool cmp(type aa,type bb)

{

return aa.ff < bb.ff; //direct

}

---------------------------------------------------------------------------------------------------------------------------------

map < key , value > mp; //different elements

multimap < key , value > mmp; //same elements

unordered\_map < key , value > u\_mp;

unordered\_multimap < key , value > um\_mp;

1. operator " = "

2. operator " [] "

3. mp.begin()

4. mp.end() // same to rbegin() and rend()

5. mp.insert(element)

6. mp.empty() (!)

7. mp.size()

8. mp.erase(it) mp.erase(it first,it last)

9. mp.clear()

10. mp.count(element)

11. mp.find(element) // != mp.end()

12. mp.lower\_bound(element) // returns iterators

13. mp.upper\_bound(element)

14. mp.swap(mp2)

----------------------------------------------------------------------------------------------------------------------------------

set < type , cmp > st;

multiset < type , cmp > mst;

unordered\_set < type , cmp > u\_st;

unordered\_multiset < type , cmp > um\_st;

1. st.begin()

2. st.end() (!)

3. st.size()

4. st.empty()

5. st.find() // != st.end()

6. st.insert(type)

7. st.erase(it) st.erase(it first,it last)

8. st.lower\_bound(element)

9. st.upper\_bound(element)

10 st.swap(st2)

11. st.clear()

12. st.count(element)

struct cmp {

bool operator()(const int &aa, const int &bb) {

return aa < bb;

}

};

----------------------------------------------------------------------------------------------------------------------------------

deque < type > dq;

1. operator " = "

2. operator " [] "

3. dq.clear()

4. dq.size()

5. dq.begin()

6. dq.end()

7. dq.erase(it) dq.erase(it first,it last)

8. dq.push\_back(element)

9. dq.pop\_back()

10. dq.push\_front(element)

11. dq.pop\_front()

12. dq.empty()

13. dq.back()

14. dq.front()

----------------------------------------------------------------------------------------------------------------------------------

IMP:

1. stoi(string) //string to integer

2. to\_string(integer) // integer to string

3. // normal :

fill(a,0);

fill(a,-1);

fill(a,0x3f); //INF

4. //bool :

fill(a,0);

fill(a,1);

5. set\_intersection(first.begin(),first.end(),second.begin(),second.end(),inserter(ans,ans.begin()));

6. // erase from back

minheap.erase(--minheap.end()); // c++ 4.3.2

minheap.erase(std::prev(minheap.end())); // c++11

7. setbase , setfill , setw , setprecision

8. //sleep std::this\_thread::sleep\_for (std::chrono::seconds(1));

----------------------------------------------------------------------------------------------------------------------------------

// knuth morris pattern

class KnuthMorrisPratt {

public:

int a[N + 1];

string pat, txt;

vector < int > pos;

void LPS()

{

int len = 0, i = 1;

a[0] = 0;

int k = pat.sz;

while (i < k)

{

if (pat[len] == pat[i]) { len++; a[i] = len; i++; }

else {

if (len) len = a[len - 1];

else { a[i] = 0; i++; }

}

}

}

void PrintLPS() {

int k = pat.sz;

for (int i = 0; i < k; i++) cout << pat[i] << " "; cout << endl;

for (int i = 0; i < k; i++) cout << a[i] << " "; cout << endl;

}

void KMP()

{

LPS();

int i = 0; //pattern

int j = 0; //text

int k1 = txt.sz, k2 = pat.sz;

while (j < k1)

{

if (pat[i] == txt[j]) { i++; j++; }

if (i == k2)

{

pos.pb(j - i);

i = a[i - 1];

}

else if (j<k1 && pat[i] != txt[j])

{

if (i) i = a[i - 1];

else j++;

}

}

}

void PrintPos() { //0-based indexing

int k = pos.sz;

for (int i = 0; i < k; i++) cout << pos[i] << " "; cout << endl;

}

};

///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

// Z algorithm

class Zalgo {

public:

int z[N+1];

vector < int > pos;

string pat,txt,A;

int k,k1,k2;

void Zarray() {

pos.clear();

A = pat + '$' + txt;

k = A.sz; k1 = txt.sz , k2 = pat.sz;

int L = 0, R = 0, tmp;

z[0] = k;

for(int i=1;i<k;i++) {

if(i > R) {

L = R = i;

while(R < k && A[R-L]==A[R]) R++;

z[i] = R-L; R--;

}

else {

tmp = i-L;

if(z[tmp] < R-i+1) z[i] = z[tmp];

else {

L = i;

while(R < k && A[R-L]==A[R]) R++;

z[i] = R-L; R--;

}

}

}

}

void PrintZ() {

for(int i=0;i<k;i++) cout << A[i] << " "; cout << endl;

for(int i=0;i<k;i++) cout << z[i] << " "; cout << endl;

}

void Zpos() {

for(int i=0;i<k;i++) {

if(z[i]==k2) pos.pb(i-k2-1);

}

}

void PrintPos() { // 0 based indexing

for(int i=0;i<pos.sz;i++) cout << pos[i] << " "; cout << endl;

}

};

///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

// Longest common subsequence

class LCS {

public:

string a, b;

int cnt[N + 1][N + 1];

int ln1, ln2;

char ans[N + 1];

int ComputeLCS() {

ln1 = a.sz, ln2 = b.sz;

for (int i = 0; i <= ln1; i++)

{

for (int j = 0; j <= ln2; j++)

{

if (i == 0 || j == 0) cnt[i][j] = 0;

else if (a[i - 1] == b[j - 1]) cnt[i][j] = cnt[i - 1][j - 1] + 1;

else cnt[i][j] = max(cnt[i - 1][j], cnt[i][j - 1]);

}

}

return cnt[ln1][ln2];

}

void PrintLCS() {

int idx = cnt[ln1][ln2];

int by = idx;

int i = ln1, j = ln2;

while (i>0 && j>0)

{

if (a[i - 1] == b[j - 1])

{

ans[idx - 1] = a[i - 1];

i--;

j--;

idx--;

}

else if (cnt[i - 1][j] > cnt[i][j - 1]) i--;

else j--;

}

cout << by << endl;

for (i = 0; i < by; i++) cout << ans[i]; cout << endl;

}

};

/////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

// TRIE

struct trie {

trie \*child[27];

bool is\_word;

trie \*new\_node() {

trie \*tmp = new trie;

for(int i=0;i<26;i++) tmp->child[i] = NULL;

tmp->is\_word = 0; return tmp;

}

void addword(trie \*root,string s) {

trie \*tmp = root;

int ch;

for(int i=0;i<s.sz;i++) {

ch = (int)(s[i]-'a');

if(tmp->child[ch]==NULL) tmp->child[ch] = new\_node();

tmp = tmp->child[ch];

}

tmp->is\_word = 1;

}

bool findword(trie \*root,string s) {

trie \*tmp = root;

int ch;

for(int i=0;i<s.sz;i++) {

ch = (int)(s[i]-'a');

if(tmp->child[ch]==NULL) return 0;

tmp = tmp->child[ch];

}

return tmp->is\_word;

}

bool isempty(trie \*node) {

for(int i=0;i<26;i++) {

if(node->child[i]!=NULL) return 0;

}

return 1;

}

bool remove\_word(trie \*root,string s,int idx,int n) {

if(idx==n) {

root->is\_word = 0;

return 1;

}

int ch = (int)(s[idx]-'a');

if(idx!=n && root->child[ch]==NULL) return 0;

if(remove\_word(root->child[ch],s,idx+1,n)) {

if(isempty(root->child[ch])) root->child[ch]=NULL;

return (!root->is\_word);

}

return 0;

}

bool print\_order(trie \*root,string s) {

if(root->is\_word) {

cout << s << endl;

}

for(int i=0;i<26;i++) {

if(root->child[i]!=NULL) print\_order(root->child[i],s+(char)(i+'a'));

}

}

};

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

// StringHashing

#define HK 2

class StringHash {

public: //5 , 13 , 89 , 233 , 1597 , 28657

ll Nhash[N+1][HK] , exp[N+1][HK];

const int BASE[2] = {233 , 1597};

const int mod[2] = {1000000000+7 , 1000000000+13};

int k; string HashS;

void HashGen() {

k = HashS.sz; for(int i=0;i<HK;i++) {Nhash[0][i] = 0; exp[0][i] = 1;}

for(int i=1;i<=k;i++) {

for(int j=0;j<HK;j++) {

Nhash[i][j] = Madd(Mmul(Nhash[i-1][j], BASE[j] , mod[j]), (int)(HashS[i-1]-'a'+1), mod[j]);

exp[i][j] = Mmul(exp[i-1][j], BASE[j], mod[j]);

}

}

}

ll get1(int L , int R) { return Msub(Nhash[R][0], Mmul(Nhash[L-1][0], exp[R-L+1][0], mod[0]) , mod[0]); }

ll get2(int L , int R) { return Msub(Nhash[R][1], Mmul(Nhash[L-1][1], exp[R-L+1][1], mod[1]) , mod[1]); }

pll get(int L, int R) { // 1-indexing

if(HK==1) return pll(get1(L,R),0);

return pll(get1(L,R),get2(L,R));

}

}A,B;

bool check(pll XX,pll YY) { return (XX.ff==YY.ff && XX.ss==YY.ss); }

int HashIndex(int L,int R) { // 1-indexing

int low = L , high = R , mid;

while(low <= high) {

mid = (low+high) >> 1;

if(check(A.get(L,mid),B.get(L,mid))) low = mid+1;

else high = mid-1;

}

if(low-1 < L) return -1;

return low-1;

}

-----------------------------------------------------------------------------------------------------------------------------------------------------

// TOPOSORT - 2 // first print all child nodes format

vector < int > adj[N];

int indegree[N];

vector < int > vi;

void toposort(int n) {

queue < int > q;

FOR(i , 1 , n) if(!indegree[i]) q.push(i);

while(!q.empty()) {

int k = q.front();

vi.pb(k);

q.pop();

for(auto i: adj[k]) {

indegree[i]--;

if(!indegree[i]) q.push(i);

}

}

if(vi.sz!=n) { cout << "Cycle" << endl; return; }

reverse(vi.begin() , vi.end());

FOR(i , 1 , n) cout << vi[i-1] << " "; cout << endl;

}

int main()

{

Nitro; openup; closeup;

int n , e;

cin >> n >> e;

FOR(i , 1 , n) indegree[i] = 0;

FOR(i , 1 , e) {

int u , v;

cin >> u >> v;

adj[u].pb(v);

indegree[v]++;

}

toposort(n);

return 0;

}

---------------------------------------------------------------------------------------------------------------------------------------------------------

// strongly connected components - kosarajju's

vector < int > adj1[N] , adj2[N];

bool ok[N];

stack < int > sk;

void toposort(int u) {

if(ok[u]) return ;

ok[u] = 1;

for(auto i : adj1[u]) toposort(i);

sk.push(u);

}

void dfs2(int u) {

if(ok[u]) return ;

ok[u] = 1;

for(auto i : adj2[u]) dfs2(i);

cout << u << " ";

}

int main()

{

Nitro; openup; //closeup;

int n , e;

cin >> n >> e;

int u , v;

for(int i=0;i<e;i++) {

cin >> u >> v;

adj1[u].pb(v);

adj2[v].pb(u);

}

FOR(i , 0 , n-1) if(!ok[i]) toposort(i);

fill(ok , 0);

//while(!sk.empty()) { cout << sk.top() << " "; sk.pop(); } cout << endl;

while(!sk.empty()) {

int u = sk.top();

sk.pop();

if(ok[u]) continue;

dfs2(u);

cout << endl;

}

return 0;

}

------------------------------------------------------------------------------------------------------------------------------------------------

// ( UNION - FIND ) & ( PATH - COMPRESSION ) (2)

int c;

int Rank[N];

int parent[N];

int dsize[N];

int assign(int m) { //path-compression

if(parent[m]==m) return m;

return parent[m]=assign(parent[m]);

}

void addsize(int x,int y) {

dsize[x] += dsize[y];

dsize[y] = 0;

}

void Merge(int x,int y) { // Union-find

int xroot=assign(x);

int yroot=assign(y);

if(xroot!=yroot) {

c--; // no of components

if(Rank[xroot] < Rank[yroot]) { parent[xroot]=yroot; addsize(yroot , xroot); }

else if(Rank[xroot] > Rank[yroot]) { parent[yroot]=xroot; addsize(xroot , yroot); }

else {

Rank[xroot]++;

parent[yroot]=xroot;

addsize(xroot , yroot);

}

}

}

int main()

{

Nitro;

int n , e;

cin >> n >> e;

FOR(i , 1 , n) {

parent[i] = i;

dsize[i] = 1;

}

c = n;

FOR(i , 1 , e) {

int u , v;

cin >> u >> v;

Merge(u , v);

}

}

---------------------------------------------------------------------------------------------------------------------------------------------------------

// MST - Prim's

#define pii pair < int , int >

vector < pair < int , pair < int , int > > > vi;

vector < pair < int , int > > adj[N];

bool ok[N];

struct cmp {

bool operator() (const pair < int , int > &xx , const pair < int , int > &yy) {

return xx.ff > yy.ff;

}

};

void primsMST() {

int mst = 0;

priority\_queue < pair < int , int > , vector < pair < int , int > > , cmp > pq;

pq.push({0 , 1}); // weight - edge format

while(!pq.empty()) {

pair < int , int > tmp = pq.top();

pq.pop();

if(ok[tmp.ss]) continue;

mst += tmp.ff;

for(auto i : adj[tmp.ss]) {

if(!ok[i.ss]) pq.push({i.ff , i.ss});

}

ok[tmp.ss] = 1;

}

cout << "mst = " << mst << endl;

}

int main()

{

Nitro; openup; //closeup;

int n , m;

cin >> n >> m;

int x , y , w;

FOR(i , 1 , m ) {

cin >> x >> y >> w;

vi.pb({w , {x , y}});

adj[x].pb({w , y});

adj[y].pb({w , x});

}

primsMST();

return 0;

}

-----------------------------------------------------------------------------------------------------------------------------------------------------

// BIT MANIPULATIONS

template <class type> class BITmanip {

public:

inline type two(type n) { return 1 << n; }

inline type test(type n, int b) { return (n>>b)&1; } // returs true or false

inline type set\_bit(type & n, int b) { n |= two(b); }

inline type unset\_bit(type & n, int b) { n &= ~two(b); }

inline type last\_bit(type n) { return n & (-n); } // returns value not place

inline type ones(type n) { int res = 0; while(n && ++res) n-=n&(-n); return res; } //no of set bits

};

BITmanip < ll > bt;

-----------------------------------------------------------------------------------------------------------------------------------------------

// NcR single calculation

ll NCR(int n , int r) {

if(n < 0 || r < 0) return -1;

if(n < r) return 0;

ll result = 1;

if(r > n/2) r = n-r;

FOR(i , 0 , r - 1) {

result \*= (n-i);

result /= (i+1);

}

return result;

}

===================================================================================

int isprime[N+2];

vector <int > prime;

void sieve() {

FOR(i , 0 , N) isprime[i] = i;

isprime[0] = 0; isprime[1] = 0;

int p = sqrt(N);

FOR(i , 2 , p) {

if(isprime[i] == i) {

for(int j = i+i;j <= N;j += i)

isprime[j] = i;

}

}

FOR(i , 2 , N) if(isprime[i] == i) prime.pb(i);

}

==================================================================================

// euler totient function for a given range

int phi[N+2];

void euler() {

phi[1] = 1;

for(int i=2; i<=N; i++) {

if(!phi[i]) {

phi[i] = i-1;

for(int j=(i<<1); j<=N; j+=i) {

if(!phi[j]) phi[j] = j;

phi[j] = phi[j]/i\*(i-1);

}

}

}

}

-------------------------------------------------------------------------------------------------------------------------------------------

// extended gcd

int gcdExtended(int a, int b, int \*x, int \*y)

{

if (a == 0)

{

\*x = 0;

\*y = 1;

return b;

}

int x1, y1;

int gcd = gcdExtended(b%a, a, &x1, &y1);

\*x = y1 - (b/a) \* x1;

\*y = x1;

return gcd;

}

----------------------------------------------------------------------------------------------------------------------------------------------

void Fibanocci(ll n, ll&x, ll&y){

if(n<=0) { x = 0; y = 1; return; }

if(n&1){ Fibanocci(n-1, y, x); y=Madd(x,y,MOD);}

else{

ll a, b;

Fibanocci(n>>1, a, b);

y = Madd(Mmul(a,a,MOD),Mmul(b,b,MOD),MOD);

x = Madd(Mmul(a,b,MOD),Mmul(a,(b-a+MOD),MOD),MOD);

}

}

---------------------------------------------------------------------------------------------------------------------------------------------

// powers

ll Madd(ll a,ll b,ll mod){ll c=(ll)((ll)(a)+(ll)(b));while(c>=mod){c-=mod;};while(c<0){c+=mod;};return c;}

ll Msub(ll a,ll b,ll mod){return add(a,mod-b,mod);}

ll Mmul(ll a,ll b,ll mod){ll c=(ll)((ll)(a)\*(ll)(b));c%=mod;while(c<0){c+=mod;};return c;}

ll Mpow(ll a,ll n,ll b){ll res=1;while(n){if(n&1) {res=mul(res,a,b);}a=mul(a,a,b);n>>=1;}return add(res,0,b);}

ll Mdiv(ll a,ll b,ll mod){ll ans=mul(a,mod\_pow(b,mod-2,mod),mod);return ans;}

---------------------------------------------------------------------------------------------------------------------------------------------

ll lucas(ll n,ll r)

{

if (n==0 && r==0) return 1;

ll ni = n%MAX;

ll ri = r%MAX;

if (ni<ri) return 0;

return ( lucas(n/MAX, r/MAX)\*smallC(ni, ri) );

}

--------------------------------------------------------------------------------------------------------------------------------------------

/\*DIJSKTRA ALGO \*/

vector <pii> V[10];

int N,E;

int dist[20];

struct cmp

{

bool operator()(const pii &x, const pii &y) // in case of pairs (const pair<int,int> &l ,...)

{

return x.ss > y.ss; // return l.ff/l.ss < r.ff/r.ss

}

};

int djs(int source,int target)

{

priority\_queue <pii,vector<pii>,cmp> pq;

fill(dist,0x3f); // assigning INF for every node in distance array

dist[source]=0; // source at 0

pq.push(mp(source,0)); // edge-weight

while(!pq.empty())

{

int u,v,w,c;

u=pq.top().ff;

c=pq.top().ss; // node cost so far

pq.pop();

if(u == target) return dist[u];

if(dist[u] < c) continue; // already visited and hav min value than current - continue

for(int i=0;i<V[u].sz;i++)

{

v=V[u][i].ff;

w=V[u][i].ss;

if(dist[v] > dist[u]+w)

{

dist[v]=dist[u]+w;

pq.push(mp(v,dist[v]));

}

}

}

return -1;

}

------------------------------------------------------------------------------------------------------------------------------------------------------

// MO

int a[305]; //by n

int score[105]={0}; //by values

int k,n;

int ans[25]; //by t

int L,R; //current pos

int res=0; //for queries

typedef struct node

{

int L,R;

int i;

}node;

bool cmp(const node &x, const node &y)

{

if((x.L/k) == (y.L/k))

return x.R < y.R;

else

return (x.L/k)<(y.L/k);

}

void ad(int l)

{

score[a[l]]++;

if(score[a[l]]==1) res++;

}

void del(int l)

{

score[a[l]]--;

if(score[a[l]]==0) res--;

}

int main()

{

// scan n;

k=(int)sqrt(n);

for(int i=0;i<n;i++)

sc(a[i]);

int t; //queries

// scan t;

node q[t+3];

// scan q;

sort(q,q+t,cmp);

L=0;

R=-1;

for(int i=0;i<t;i++)

{

while(L<q[i].L)

{

del(L);

L++;

}

while(L>q[i].L)

{

L--;

ad(L);

}

while(R<q[i].R)

{

R++;

ad(R);

}

while(R>q[i].R)

{

del(R);

R--;

}

ans[q[i].i]=res;

}

}

-------------------------------------------------------------------------------------------------------------------------------------------------

// BIT

template <class type> class Fenwick {

public:

type tree[N+1];

int n;

type get(int idx) {

type sum = 0;

while(idx) {

sum += tree[idx];

idx -= (idx&-idx);

}

return sum;

}

void update(int idx,type val) {

while(idx <= n) {

tree[idx] += val;

idx += (idx&-idx);

}

}

void init(int n) {

this->n = n;

for(int i=0;i<=n;i++) tree[i]=0;

}

};

----------------------------------------------------------------------------------------------------------------------------------

//Segmented tree

template <class type> class SegmentTree {

public:

type st[4 \* N] , a[N];

void init() {

for(int i=0;i<4\*N;i++) st[i] = 0;

for(int i=0;i<N;i++) a[i] = 0;

}

type merge(type x, type y) {

return x+y;

}

type Sborder() { return 0; }

type Scondition(int idx) { return st[idx]; }

type Uborder(int idx) { return st[idx]; }

type Ucondition(int idx, int val) { return st[idx] + val; }

type search(int qs,int qe,int s,int e,int idx)

{

if(qe < qs || e < s) return Sborder();

int mid=(s+e)/2; type b;

if(s>qe || e<qs) return Sborder();

if(s>=qs && e<=qe) return Scondition(idx);

return b = merge(search(qs,qe,s,mid,idx\*2+1), search(qs,qe,mid+1,e,idx\*2+2));

}

type construct(int s,int e,int idx)

{

int mid=(s+e)/2;

if(s==e) return st[idx] = a[s];

return st[idx] = merge(construct(s,mid,idx\*2+1), construct(mid+1,e,idx\*2+2));

}

type update(int us,int ue,int s,int e,int idx,type val)

{

int mid=(s+e)/2;

if(s>ue || e<us) return Uborder(idx);

if(s==e && s>=us && e<=ue) return st[idx] = Ucondition(idx, val);

return st[idx] = merge(update(us,ue,s,mid,idx\*2+1,val), update(us,ue,mid+1,e,idx\*2+2,val));

}

};

----------------------------------------------------------------------------------------------------------------------------------

//Lazy propagation

template <class type> class LazyTree {

public:

type st[4 \* N];

type lazy[4 \* N];

type a[N];

void init() {

for(int i=0;i<4\*N;i++) {

lazy[i] = 0;

st[i] = 0;

}

for(int i=0;i<N;i++) a[i] = 0;

}

type merge(type x, type y) {

return x+y;

}

type Sborder() { return 0; }

type Scondition(int idx) { return st[idx]; }

type Uborder(int idx) { return st[idx]; }

type Ucondition(int idx) { return st[idx]; }

void LazyUpdate(int s,int e,int idx,int val) {

st[idx] += (e-s+1)\*val;

if(s!=e) {

lazy[idx\*2+1] += val;

lazy[idx\*2+2] += val;

}

lazy[idx] = 0;

}

type search(int qs,int qe,int s,int e,int idx)

{

int mid=(s+e)/2; type b;

if(lazy[idx]) {

LazyUpdate(s,e,idx,lazy[idx]);

}

if(s>qe || e<qs) return Sborder();

if(s>=qs && e<=qe) return Scondition(idx);

return b = merge(search(qs,qe,s,mid,(idx\*2)+1), search(qs,qe,mid+1,e,(idx\*2)+2));

}

type construct(int s,int e,int idx)

{

int mid=(s+e)/2;

if(s==e) return st[idx] = a[s];

return st[idx] = merge(construct(s,mid,(idx\*2)+1), construct(mid+1,e,(idx\*2)+2));

}

type update(int us,int ue,int s,int e,int idx,type val)

{

int mid=(s+e)/2;

if(lazy[idx]) {

LazyUpdate(s,e,idx,lazy[idx]);

}

if(s>ue || e<us) return Uborder(idx);

if(s>=us && e<=ue) {

LazyUpdate(s,e,idx,val);

return st[idx] = Ucondition(idx);

}

return st[idx] = merge(update(us,ue,s,mid,(idx\*2)+1,val), update(us,ue,mid+1,e,(idx\*2)+2,val));

}

};

-------------------------------------------------------------------------------------------------------------------------------------------------------

// next permutation

unsigned int t = v | (v - 1);

w = (t + 1) | (((~t & -~t) - 1) >> (\_\_builtin\_ctz(v) + 1));

unsigned int t = (v | (v - 1)) + 1;

w = t | ((((t & -t) / (v & -v)) >> 1) - 1);

-------------------------------------------------------------------------------------------------------------------------------------------------------

Input - 1 based indexing i.e (lo = 1 , hi = n)

Output - 0 based indexing i.e (0 to n - 1)

int Bsearch(int lo,int hi,int req) {

int mid;

while(lo <= hi) {

mid = lo + ((hi-lo)>>1);

if(a[mid-1]==req) return mid-1;

if(a[mid-1] > req) hi = mid-1;

else lo = mid+1;

} return -1;

}

int UpperSearch(int lo,int hi,int req) {

int mid;

while(lo <= hi) {

mid = lo + ((hi-lo) >> 1);

if(a[mid-1] <= req) lo = mid + 1;

else hi = mid - 1;

} return hi;

}

int LowerSearch(int lo,int hi,int req) {

int mid;

while(lo <= hi) {

mid = lo + ((hi - lo) >> 1);

if(a[mid-1] >= req) hi = mid - 1;

else lo = mid + 1;

} return hi;

}

-----------------------------------------------------------------------------------------------------------------------------------------------------------

inline int diff(double lhs, double rhs) {

if (lhs - eps < rhs && rhs < lhs + eps) return 0;

return (lhs < rhs) ? -1: 1;

}

struct Point {

double x, y;

Point() {}

Point(double aa, double bb) : x(aa), y(bb) {}

bool operator == (const Point &rhs) const {

return ((diff(x, rhs.x) == 0) && (diff(y, rhs.y) == 0));

}

const Point operator + (const Point &rhs) const {

return Point(x + rhs.x, y + rhs.y);

}

const Point operator - (const Point &rhs) const {

return Point(x - rhs.x, y - rhs.y);

}

const Point operator \* (double t) const {

return Point((double)(x)\*(double)(t), (double)(y)\*(double)(t));

}

};

inline double Slope(const Point &pp, const Point &qq) {

return (qq.x - pp.x) ? (double)(qq.y - pp.y) / (double)(qq.x - pp.x) : INF;

}

struct Circle {

Point center;

double r;

Circle() {}

Circle(const Point &aa, double bb) : center(aa), r(bb) {}

};

struct Line {

Point s, d;

double a,b,c,m;

Line() {}

Line(const Point &aa, const Point &bb) {

s = aa; d = bb; m = Slope(s,d);

a = d.y - s.y; b = s.x - d.x;

c = (s.y\*(d.x-s.x)) - (s.x\*(d.y-s.y));

}

};

inline double DOT(const Point &pp, const Point &qq) {

return (double)(pp.x)\*(double)(qq.x) + (double)(pp.y)\*(double)(qq.y);

}

inline double CROSS(const Point &pp, const Point &qq) {

return (double)(pp.x)\*(double)(qq.y) - (double)(pp.y)\*(double)(qq.x);

}

inline bool OnLine(const Point &pp, const Point &rr, const Point &qq) {

return diff(CROSS(qq - pp, rr - pp), 0) == 0;

}

inline bool IsBetween(const double &pp, const double &rr, const double &qq) {

return (diff(qq , max(pp, rr))<=0) && (diff(qq, min(pp , rr))>=0);

}

inline bool OnSegment(const Point &pp, const Point &rr, const Point &qq) {

return OnLine(pp, rr, qq) && IsBetween(pp.x, rr.x, qq.x) && IsBetween(pp.y, rr.y, qq.y);

}

inline double dist(const Point &pp, const Point &qq) {

return double(sqrt(DOT(pp - qq, pp - qq)));

}

Point LineIntersection(const Line &pp,const Line &qq) {

double a1, a2;

a1 = (((pp.b\*qq.c)-(qq.b\*pp.c))/((pp.a\*qq.b)-(qq.a\*pp.b)));

a2 = (((pp.c\*qq.a)-(qq.c\*pp.a))/((pp.a\*qq.b)-(qq.a\*pp.b)));

return Point(a1,a2);

}

double AreaOfPolygon(vector < Point > l, int n) {

double pp = 0 ,qq = 0;

for(int i=0;i<n;i++) {

pp += (double)(l[i].x)\*(double)(l[(i+1)%n].y);

qq += (double)(l[(i+1)%n].x)\*(double)(l[i].y);

}

double ans = (pp-qq); ans = abs(ans); ans/=2;

return ans;

}

double AngleBetweenLines(const Line &pp , const Line &qq) {

double CA = atan2(pp.d.y-pp.s.y , pp.d.x-pp.s.x); // tc with directions

double CB = atan2(qq.d.y-qq.s.y , qq.d.x-qq.s.x);

double c\_ang = CA-CB; if(c\_ang > PI) c\_ang -= 2\*PI;

if(c\_ang < -PI) c\_ang += 2\*PI; // result - [-PI , PI]

c\_ang \*= (ld)(180)/(ld)(PI);

if(diff(c\_ang , 360.0) >= 0) c\_ang-=360.0;

if(diff(c\_ang , 0.0) < 0) c\_ang+=360.0;

return c\_ang; // result clockwise and [0 , 360)

}

bool IsPointInsidePolygon(Point p ,vector <Point> pts ,int n) {

for(int i=0;i<n;i++) { if(OnSegment(pts[i],pts[(i+1)%n],p)) { return 1; } } // on polygon

int c1 = 0; double x1 , x2 , k , m , y2 , by;

for(int i=0;i<n;i++) {

if(diff(pts[i].x , pts[(i+1)%n].x) < 0) {

x1 = pts[i].x;

x2 = pts[(i+1)%n].x;

}

else { x1 = pts[(i+1)%n].x; x2 = pts[i].x; }

if(diff(p.x, x1) > 0 && diff(p.x, x2) <= 0 && (diff(p.y, pts[i].y) < 0 || diff(p.y, pts[(i+1)%n].y) <= 0)) {

Point d = pts[(i+1)%n] - pts[i];

if(diff(d.x , d,y)==0) k = INF;

else k = dy/dx;

m = pts[i].ss - k \* pts[i].ff;

y2 = k \* p.ff + m;

by = (double)(p.ss);

if(diff(by , y2) <= 0) c1++;

}

}

return c1&1;

}